Energy Monitoring as an Essential Building Block Towards Sustainable Ultrascale Systems

Francisco Almeida, Marcos Dias de Assuncao, Jorge Barbosa, Vicente Blanco, Ivona Brandic, Georges da Costa, Manuel F Dolz, Anne C Elster, Mateusz Jarus, Helen Karatza, et al.

Energy Monitoring as an Essential Building Block Towards Sustainable Ultrascale Systems

Francisco Almeida[a], Marcos D. Assunção[b,*], Jorge Barbosa[c], Vicente Blanco[a], Ivona Brandic[d], Georges Da Costa[e], Manuel F. Dolz[f], Anne C. Elster[g], Mateusz Jarus[h], Helen D. Karatza[i], Laurent Lefèvre[b], Ilias Mavridis[i], Ariel Oleksiak[h], Anne-Cécile Orgerie[j], Jean-Marc Pierson[e]

[a] *Universidad de La Laguna, Spain*
[b] *Inria, Ecole Normale Superieure de Lyon, France*
[c] *Universidade do Porto, Portugal*
[d] *Vienna University of Technology, Austria*
[e] *Institute of Computer Science Research, University of Toulouse, France*
[f] *Universidad Carlos III de Madrid, Spain*
[g] *Norwegian University of Science and Technology, Norway*
[h] *Poznan Supercomputing and Networking Center, Poznan University of Technology, Poland*
[i] *Department of Informatics, Aristotle University of Thessaloniki, Greece*
[j] *CNRS, IRISA, France*

Abstract

An Ultrascale System (USS) joins parallel and distributed computing systems that will be two to three orders of magnitude larger than today's infrastructure regarding scale, performance, the number of components and their complexity. For such systems to become a reality, however, advances must be made in High Performance Computing (HPC), large-scale distributed systems, and big data solutions, also tackling challenges such as improving the energy efficiency of the IT infrastructure. Monitoring the power consumed by underlying IT resources is essential towards optimising the manner IT resources are used and hence improve the sustainability of such systems. Nevertheless, monitoring the energy consumed by USSs is a challenging endeavour as the system can comprise thousands of heterogeneous server resources spanning multiple data centres. Moreover, the amount of monitoring data, its gathering, and processing, should never become a bottleneck nor profoundly impact the energy efficiency of the overall system. This work surveys state of the art on energy monitoring of large-scale systems and methodologies for monitoring the power consumed by large systems and discusses some of the challenges to be addressed towards monitoring and improving the energy efficiency of USSs. Next, we present efforts made on designing monitoring solutions[☆]. Finally, we discuss potential gaps in existing solutions when tackling emerging large-scale monitoring scenarios and present some directions for future research on the topic.

Keywords: Ultra large-scale systems, energy-awareness, energy-efficiency, monitoring

1. Introduction

Society is increasingly becoming more instrumented and hence generating large volumes of data. The data results from business processes, advanced scientific experiments, operational monitoring of various types of infrastructures, medical applications designed to assess the health conditions of populations, financial analysis, among other sources. Much of the produced data must be preserved and requires processing of some sort, which poses challenges to existing IT infrastructure concerning storage, management, interoperability, governance, processing, and analysis [1].

In this article, we focus on Ultrascale Systems (USSs), one of the envisioned solutions for handling novel applications and services that demand scores of resources. An USS joins parallel and distributed computing systems that look at Exascale [2] and beyond, which will be two to three orders of magnitude larger than today's infrastructures regarding scale, performance, the number of components and their com-

[☆] Some of the work has been conducted in the context of the Energy Efficiency working group of the Nesus European COST Action IC1305: http://www.cost.eu/COST_Actions/ict/IC1305

[*] Corresponding author

plexity. For such systems to become a reality, however, advances must be made in High Performance Computing (HPC), large-scale distributed systems, and big data solutions [3], also tackling challenges such as improving the energy efficiency of the underlying IT and cooling infrastructure.

In addition to optimising the production and lifecycle of employed equipment, improving the energy efficiency of USSs requires means to exploit the available resources efficiently, thus achieving energy proportionality [4]. Over the years, several technical solutions have been proposed to improve the efficiency of processors [5, 6], data storage solutions and network equipment [7]. Although all these techniques are crucial towards improving the overall effectiveness of USS, the amount of energy consumed by large-scale distributed systems is still heavily impacted by how resources are allocated to applications. Such allocation decisions often carried out by resource management systems [8], should be made considering how much energy the underlying resources consume.

Monitoring is therefore essential for determining how much power the resources consume and the impact of allocation decisions on the system overall energy efficiency. However, monitoring the energy consumption of USSs is a challenging endeavour as the system can comprise thousands of heterogeneous server resources spanning multiple data centres. Moreover, the amount of monitoring data, its gathering and processing, should not heavily impact the energy efficiency of the observed system.

In this work, we survey state of the art on energy monitoring of large-scale systems and discuss some of the needs and challenges that must be addressed towards monitoring and improving the energy efficiency of USSs. The work describes energy-monitoring solutions and discusses the challenges they aim to address. We then carry out a gap analysis, discuss some challenges that are currently left unaddressed by existing solutions, and describe some areas that deserve particular attention on monitoring the energy consumption and improving the efficiency of ultra large-scale systems.

The rest of this paper is organised as follows. Section 2 surveys state of the art on power monitoring of large systems and monitoring methodologies. In Section 3, we describe a non-exhaustive list of challenges on energy monitoring for USSs. A description of selected solutions is provided in Section 4. Section 5 presents a gap analysis and discusses future directions on monitoring the energy consumed by large-scale systems. Finally, Section 6 concludes the paper.

2. Power Monitoring of Large-Scale Systems

Improving the energy efficiency of USSs is a challenging endeavour that requires advances in several areas [9]. As illustrated in Figure 1, reducing the energy consumption of USSs requires innovations in hardware and on the assignment and execution of applications and services onto available resources while taking advantage of the heterogeneous hardware. It has also been identified that to improve energy efficiency, accurate and real-time power monitoring may be required, which is difficult to achieve since, at large scale, the monitoring system can become a USS itself.

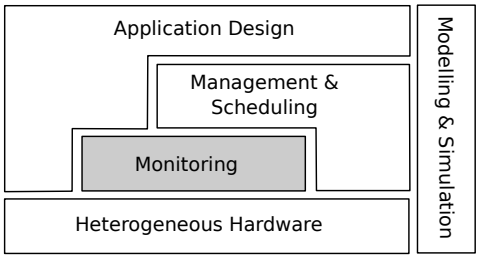

Figure 1: Areas that affect the analysis of energy efficiency in USSs.

It is desirable to gain insights into the consumption of existing systems at all possible levels to optimise the energy usage across the whole stack of a USS. System designers and operators, as well as application developers, should be able to access power data at multiple granularity levels, from the whole system to individual server components. Given the importance that monitoring has on optimising systems for energy efficiency, during the second phase of the action we opted for reviewing state of the art in this domain and describe the solutions designed by participants for monitoring the power consumed by large-scale systems.

Previous work has discussed performance analysis methodologies that consider available energy monitoring and energy awareness mechanisms for HPC architectures. Benedict [10] for instance, defines a taxonomy encompassing methods based on measurement, estimation, and analytical modelling. The author classifies energy monitoring and analysis solutions for HPC in hardware-based, software-focused and hybrid. Noureddine *et al.* surveyed approaches on energy modelling and measurement [11]. A study more focused on energy monitoring has been performed by Hsu and Poole [12], whereas Hackenberg *et al.* [13] investigated hardware-based methods to monitor the energy consumed by computing nodes. Other general surveys on energy-efficient resource allocation mechanisms have been presented [5, 14] where many of the surveyed mechanisms

Table 1: Wattmeter infrastructure.

Device Name	Interface	Refresh Time (s)	Precision (W)
Dell iDrac6	IPMI / Ethernet	5	7
Eaton	Serial, SNMP Ethernet	5	1
OmegaWatt	IrDA Serial	1	0.125
Schleifenbauer	SNMP via Ethernet	3	0.1
Watts Up?	Proprietary via USB	1	0.1
ZES LMG450	Serial	0.05	0.01

require monitoring of some type. The rest of this section provides a non-exhaustive description of technologies, approaches, and methodologies for energy monitoring of large-scale distributed systems and HPC infrastructure.

2.1. Energy Monitoring Infrastructure

Several hardware and software solutions have been used to monitor the power consumed by computing systems and data centres. Such solutions enable multiple levels of measurement with varying degrees of precision and intrusiveness. While focusing on data centres and compute clusters, we have previously classified such solutions as external devices, intra-resource devices, hardware sensors, and software interfaces (Figure 2). A revised summary of these solutions is presented here, centred on hardware and hybrid approaches, whereas a more detailed discussion is found in previous work [9].

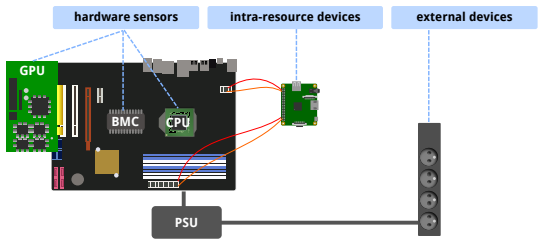

Figure 2: Energy monitoring infrastructure.

Moreover, Table 1 lists equipment deployed and used by the solutions described later in Section 4.

2.1.1. External Devices

These devices, commonly called wattmeters or powermeters, often comprise equipment not embedded into the resources whose power consumption is measured. They mostly lie in between the wall socket and the plug of the measured equipment, thus measuring the consumption of an entire sub-system (*e.g.* a server node, a storage array). Examples of this type of equipment include powermeters such as Kill-a-Watt [15], Watt's Up Pro [16], enclosure PDUs (ePDUs) with measurement and managing capabilities[1], PowerPack [17], PowerScope [18], among other solutions.

Wattmeters available in the market vary regarding physical interconnection, communication protocols, packaging and precision of measurements they take. They are mostly packaged in multiple outlet power strips called Power Distribution Units (PDUs) or ePDUs, and more recently in the Intelligent Platform Management Interface (IPMI) cards embedded in the servers themselves. Moreover, wattmeters may differ in the manner they operate; some equipment sends measurements to a management node on a regular basis (push mode), whereas others respond to queries (pull mode). Other characteristics that differ across wattmeters include:

- refresh rate (*i.e.* maximum number of measurements per second);
- measurement precision; and
- methodology applied to each measurement (*e.g.* mean of several measurements, instantaneous values, and exponential moving averages).

Depending on the measurement hardware and infrastructure, biases such as the overhead of the monitoring software, PSU conversion impact, among other issues, have to be taken into account using, for instance, the methodology described in previous work [19].

2.1.2. Intra-Resource Devices

This class of devices comprises equipment that is placed inside server nodes – often between the power supply and the main board – to measure the power consumption of individual equipment or voltage lines. Examples include the PowerMon devices [20], placed between a node's power supply and mainboard; the PowerInsight [21], designed for component-level instrumentation of commodity hardware; the ARM Energy Probe [22], integrated with the ARM development tool chain; the Linux Energy Attribution and Accounting Platform LEA^{2}P [23]; among other solutions. The High Definition Energy Efficiency Monitoring (HDEEM) infrastructure [24] which proposes a sophisticated approach towards system-wide and fine-grained power measurements by introducing, among other features, a

[1] http://www.eaton.com/Eaton/index.htm

Field-Programmable Gate Array (FPGA) on each server blade able to improve spatial granularity to measure blade, CPU, and DRAM power consumption while improving temporal granularity to up to 1 kSa/s.

2.1.3. Hardware Sensors and Software Interfaces

More recent data-centre hardware offers many built-in sensors to report the energy that a piece of equipment consumes. The data can be made available to monitoring tools via performance counters or vendor-specific APIs. Examples include the Intel Running Average Power Limit (RAPL) interface which reports per-package estimates of total energy consumed on Intel Sandy Bridge CPUs and later; the NVIDIA Management Library (NVML) interface that queries instant power drawn by NVIDIA Tesla GPUs; several motherboards report power drawn by extending IPMI.

Hardware Performance Counters (PCs) are special-purpose registers built into modern microprocessors to store the counts of hardware-related activities. Their content can be made available at the operating system level via special file descriptors available through, for instance, the Linux kernel API [25]. PCs can provide CPU information, such as clock cycles, instructions, cache references and misses, branch instructions and misses, page faults, context switches, among others. Along with power usage metrics, this information has been exploited for determining the energy cost of certain hardware operations.

Several attempts have been made over the past years towards providing standard APIs for monitoring the energy consumed by equipment in a data centre or HPC. Reviewed in previous work, some of these initiatives are the Performance API (PAPI) [26], which includes an integrated energy consumption interface for monitoring the energy consumed by computing resources; the PowerAPI, an attempt to standardise access to power measurement data and power control [27]; the Energy Measurement Library (EML), a software library that simplifies the analysis of energy consumed by heterogeneous systems; among other power measurement APIs [28]. Concerning HDEEM infrastructure mentioned beforehand, a particular API has been designed to access power and energy measurements supporting both in-band access through GPIO and PCIe as well as out-of-band access through IPMI [29]. Besides, at the level of resource and job management systems for HPC, functionalities have been introduced [30] to directly relate the monitoring measurements to power profiling and energy accounting per job. These features facilitate the analysis of energy data by users and administrators during the real-scale executions upon the HPC platforms.

2.2. Monitoring Methodologies

The power-monitoring methodology proposed by the Standard Performance Evaluation Corporation (SPEC) [31] is certainly one of the most popular methodologies for evaluating the energy consumed by a system under test. It was designed for evaluating the energy efficiency of a server or a group of servers running a Java-based business benchmark. Although the methodology was initially designed for server workloads, it has been adapted to suit other scenarios [32] as highlighted by Scogland *et al.* [33].

Some initiatives have been undertaken towards devising methodologies for monitoring the energy consumption of large computing systems and for enabling architectural analysis and comparison for rankings such as the Top500 [34] and Green500 [35]. One important effort has been made by the Energy Efficient HPC Working Group (EE HPC WG) [33], which by undertaking a survey of power submissions to the Green500 and Top500 lists, demonstrated that there is a wide variation in the quality of the measurements reported. Though some of the analysed power submissions were comprehensive and reflected a high level of quality, others were based on course grained information such as specification sheets provided by hardware vendors. Aiming to provide a common methodology the EE HPC WG identified several issues, including:

- Unclear demarcation between the computer system and the data centre infrastructure, *e.g.* fans, power supplies, and liquid cooling.
- Use of shared resources such as storage and networking.
- Limitations on data centre and instrumentation for system level power measurement.

To accommodate systems that cannot be fully instrumented, the provided methodology proposed three levels of measurements. Level 1 is similar to Green500's Rev0.9 rules, where a single average power measurement is extrapolated from a machine subset. Level 2 still consists in a power measurement based on a subset of the overall system. Level 3, the most rigorous, offers a valid, accurate measurement of the full system, but it is often only possible at a few sites. The methodology then defines three quality levels, namely *good*, *better*, and *best* with Level 3 being the best. The quality ratings impose requirements on four different aspects of power measurement as follows:

- The time span over which a measurement is taken, the time granularity, and the reported measurements.
- The number of resources, or fraction of the system that is instrumented.
- Subsystems required in the measurement.
- Where in the power system the measurements are taken.

The EE HPC methodology distinguishes between the core phase of a workload and the entire workload, while taking the Green500 workload as a basis. Table 2 summarises the various aspects considered by the methodology and the criteria used by each proposed level and quality. This methodology was later applied to investigate the inter-node power usage variability [36] as explained next.

3. Challenges in Energy Monitoring and Usage

As measuring the power consumed by a large-scale system is often difficult, many published measurements are extrapolated. Previous work has investigated the validity of extrapolations in the context of inter-node power variability [37] and power variations over time within a workload run [36]. By characterising the power variability across nodes at eight supercomputer centres, a study showed that the current requirement for measurements submitted to the Green500 [35] and others allow for variations of up to 20% due to measurement timing and 10-15% due to insufficient sample sizes [36]. Other works have determined that high sampling rates are required to capture power fluctuations, while wattmeters have to be accurate and reliable enough to assess the energy efficiency of applications [38]. Although challenging, these analyses demonstrate the need for:

- high-quality, fine-grained levels of measurement;
- a methodology for measuring the power consumed by a large-scale system and reporting results; and
- transparent and verifiable means for comparing energy-efficiency results obtained from different systems.

Moreover, characterising the energy consumed by software components may require monitoring their resource utilisation and impact on the overall system consumption. Such characterisations may need monitoring both configuration and performance of multiple elements, or levels, of the software stack. As we envision that USSs may not only be supercomputers, but also federations of data centres, edge computing systems [39], and surrounding infrastructure, providing fine-grained measurements of power consumption can be challenging. Significant reduction of the energy consumed by such complex, large-scale systems also requires addressing many challenges. Some of the problems regarding techniques and resulting data management for large systems are as follows:

C1: Large-scale monitoring and profiling. To understand how resources of a large system consume energy, power consumption must be adequately measured, and information collected, integrated and analysed. For understanding how a large-scale software system consumes energy, not only the energy consumed by the physical resources should be monitored, but also how services and applications make use of the underlying resources. Proposed architectures for such monitoring systems should be capable of handling large amounts of data without creating bottlenecks. Ideally, architecture should enable, through carefully designed communication and interfaces, accessing fine-grained measurement data when and where needed. Since monitoring and profiling can be performed at multiple levels, a significant research challenge is to provide monitoring for multiple layers of software stack and techniques for propagation of metrics between the layers. Some of the challenging endeavours towards this end include:

1. **Fine-grained energy monitoring:** Fine-grained measurement here are approaches that, at a large-scale, provide or exceed Level 2 of the EEHPC methodology described earlier [33] where measurements are taken on a per-second basis at the upstream power conversion of at least 1/8 of the computing subsystem. Previous work has shown that such a granularity can help reduce variations across energy reports [36]. Monitoring systems that comply with or exceed such level are here considered to be fine-grained. Data management under fine-grained monitoring for large systems becomes an issue due to the amount of data that needs to be transferred over the network, processed and stored.
2. **Application and service monitoring:** Characterising the energy consumption of a software subsystem generally requires – in addition to measuring the energy the hardware consumes – monitoring performance and resource utilisation metrics

Table 2: Summary of aspects and quality levels [33].

Aspect	Level 1	Level 2	Level 3
Granularity	One power sample per second	One power sample per second	Continuously integrated energy
Timing	The longer of one minute or 20 % of the run	Equally spaced across the full run	Equally spaced across the full run
Measurements	Core phase average power	10 average power measurements in the core phase; Full run average power; Idle power	10 average power measurements in the core phase; Full run average power; Idle power
Machine Fraction	The greater of 1/64 of the compute subsystem or 1 kW	The greater of 1/8 of the compute subsystem or 10 kW	All included subsystems
Subsystems	Compute-nodes only	All participating subsystems, either measured or estimated	All participating subsystems must be measured
Point of Measurement	Upstream of power conversion **or** Conversion loss modelled with manufacturer data	Upstream of power conversion **or** Conversion loss modelled with off-line measurements of a single power supply	Upstream of power conversion **or** Conversion loss measured simultaneously

or counters. The underlying challenge consists in providing techniques for monitoring resource utilisation and instrumenting applications or services that can scale without causing significant overheads in the instrumented system.

3. **Resource and service profiling:** Monitoring solutions are commonly used to accumulate time series of measured metrics that can be used to build usage profiles. Concerning energy consumption, the monitoring data is utilised to create resource- and service-level profiles. While the former requires monitoring the energy consumed by the physical resources, the latter often demands correlating energy consumption with resource usage metrics.
4. **Scalable architectures:** To cope with the amount of data generated by monitoring systems at large-scale, architectures should provide means for achieving horizontal scalability, elasticity, and potentially exploit specialised hardware (*e.g.* FPGAs) that can reduce the delay of gathering data and making it available to Application Programming Interfaces (APIs) and visualisation interfaces.

C2: Models and simulation of energy consumption. As measuring the energy consumed by all components of USS could cause communication bottlenecks and high additional costs, accurate estimation methods are useful when a fine-grained measurement is costly. To this end, precise yet computationally simple modelling of USS components is needed. For the same reason as above, measurement methods that allow reducing the number of sensors would be necessary. These methods could include disaggregation techniques to retrieve fine-grained information from coarse-grained measurement data. In a USS, energy monitoring will have to be accompanied by accurate models and estimation methods that are required to reduce the number of sensors and meters, and consequently, cost and complexity of such systems. Moreover, there is also a need for incorporating power-consumption models in cloud simulators, so that researchers can evaluate software solutions for energy-efficient resource allocation.

1. **Models and estimations:** Monitoring all system components is not always possible. A tool should provide models and means to estimate the energy consumed by individual subsystems. While building such models in a small and controlled environment already presents several challenges, devising models that account for factors such as internode variability and that can be tested and validated at large-scale is a difficult endeavour [40].
2. **Integration with simulation tools:** Monitoring tools are often used to gather the data required to devise energy consumption models that are incorporated into simulation tools. Simulation tools are in turn commonly used to extrapolate certain scenarios and investigate application scheduling and resource management algorithms at scale.

C3: APIs and developer/user feedback. A large part of the progress on energy efficiency has been made in the hardware itself by for instance adopting low-power CPUs or accelerators such as GPUs and FPGAs. As large-scale systems are often very heterogeneous – and heterogeneity is likely to increase at ultrascale – software developers and users often want to know what

resources deliver the best performance to energy consumption ratio for their workload. However, as highlighted in previous work, it is challenging to design a power-monitoring infrastructure that can provide timely and accurate feedback to system developers and application writers so that they can optimise the use of energy [41]. Many monitoring infrastructures are not designed with application-level power consumption measurements in mind. Yoshii *et al.* [41] investigated the existing power monitoring of IBM Blue Gene systems and found that meaningful power consumption data can be obtained under Blue Gene/P, where the measurement interval is in the order of minutes, by carefully choosing micro-benchmarks. Blue Gene/Q, which offers per second-scale resolution of power data, allows for studying the power characteristics of the Floating Point Unit (FPU) and memory subsystems. The authors of the study argue that improvements can be made if power data were released more frequently. However, when increasing the frequency of measurements, the issue of jitter arises. The bus and FPGA speed pose limits on the sampling rate, which may impact the energy efficiency of the overall system and compromise precision. One of the proposed solutions is to enable applications to specify the start and end of monitoring intervals with the sampling and integration handled in hardware.

1. **APIs:** Users who want to assess the energy consumed by an infrastructure or its software subsystems require APIs to interface with the monitoring system to obtain data on energy consumption and configure parameters such as monitoring intervals, measurement frequency, among other factors. Such APIs should be able to cope with large amounts of data while remaining responsive.
2. **Interactive graphical interfaces:** Visualisation tools that can deal with large quantities of data, while considering their quality and presentation to facilitate navigation, are increasingly important [42]. The type of visualisation may have to be adjusted dynamically according to the amount of data to be displayed to improve both displaying and performance. There has been an effort to explore visualisation in complex data analyses by using, for instance, sophisticated reports and storytelling [43]. Fisher *et al.* [44] point out, however, that many platforms that process large amounts of data still resemble the *batch-job* model where users typically submit their jobs and wait until the execution is complete to download and analyse results to validate full runs. The authors issue a call to arms for both research and development of better interactive interfaces for data analysis where users iteratively pose queries and see rapid responses. Energy monitoring and visualisation at ultrascale are likely to face similar challenges.

C4: Integrating other measurements in architectures and APIs. Power usage is not the only significant measurement value for energy efficiency improvements, especially in the case of large-scale systems. The overall energy consumption also depends on environmental parameters such as room temperature and outside temperature, air/liquid flow, humidity, cooling system power usage, availability of renewable energy, etc. Hence, effective integration of these measurements is an important part of USS monitoring.

4. Solutions in the Scope of the Nesus Action

This section presents solutions proposed in the area of energy monitoring of large-scale computing systems[2]. We describe the solutions – namely KWAPI, EML, PMLib, ECTools and BEMOS – their design goals, challenges they aim to address, and lessons learnt during their deployment and use.

4.1. KiloWatt API (KWAPI)

The KWAPI framework[3] was created to enable measuring the power consumed by cluster resources in OpenStack [45]. It has been adapted to support large-scale federations of clusters [46] and is currently the solution deployed on the Grid5000 platform [47] to monitor the energy consumed by computing and network resources. Its architecture, depicted in Figure 3, relies on a layer of *drivers* that retrieve power measurements from several devices, and *plug-ins* or *data consumers* that collect and process the measurement data. A bus, the *forwarder*, is responsible for the communication between these two layers.

The driver layer is controlled by a *Driver Manager* that reads a configuration file compliant with the OpenStack format and starts a thread for each entry. An entry contains a list of probes and parameters including IP address and port, the type of driver to use, and relevant metrics to measure (*e.g.* SNMP OID). The measurements that a driver obtains are represented as JavaScript

[2]The solutions described here have been conducted in the context of the Energy Efficiency working group of the Nesus European COST Action IC1305: http://www.cost.eu/COST_Actions/ict/IC1305

[3]KWAPI was originally designed during the French FSN XLCloud project: http://xlcloud.org

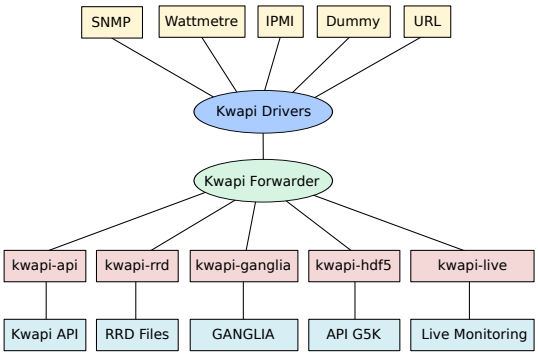

Figure 3: KWAPI architecture [46].

Object Notation (JSON) dictionaries that maintain a small footprint and are easily parsed. The size of dictionaries varies depending on the number of configured fields (*e.g.* whether messages are signed). Each driver retrieves and pushes measurements to the forwarder via ZeroMQ [48]. Drivers can manage incidents themselves, but the manager also checks periodically if all threads are active, restarting them if necessary. Support for several types of wattmeters, listed in Table 1, has been provided.

The *Forwarder* is an internal communication bus based on ZeroMQ and works as a publish/subscribe service, where drivers are publishers and plug-ins, are subscribers. It can work locally, *i.e.* with publishers and subscribers on the same machine, or through a distributed architecture using gateway devices to interconnect remote networks. Under a distributed architecture, a plug-in can listen to several drivers located at remote locations using the forwarder.

A *Data Consumer* or plug-in retrieves and processes measurements from the *Forwarder*. Two plug-ins were implemented, namely, a REST API (used to interface with Ceilometer[4]) that allows an external system to access real-time measurements, and a visualisation plug-in based on Round-Robin Database (RRD) files that expose metrics through a Web interface. Data consumers can also subscribe to receive information from drivers from multiple sites. By using a system of prefixes, consumers can subscribe to all producers or a subset. When receiving a message, a consumer verifies the signature, extracts the content and processes the data. Both drivers and consumers can be easily extended to support, respectively, several types of wattmeters and provide additional data processing services.

The current REST API allows an external system to retrieve the name of probes, measurements in Watts or kWh, and timestamps. It is secured by OpenStack Keystone tokens[5], whereby the consumer needs to ensure the validity of a token before sending a response to the system. The visualisation consumer builds RRD files from received measurements and generates graphs that show the energy consumption over a given period, with additional information such as average electricity consumption, minimum and maximum Watt values, last value, total energy and cost in Euros. More details about the visualisation features are available in previous work [45].

4.1.1. Design Goals

KWAPI has been designed to provide experimenters with deep insight into the effects of their experiments on testbeds, and users of cloud infrastructure with feedback on the energy consumed by cluster resources. It has been devised to interface with several types of sensor technologies, enabling users to specify which resources are monitored, the start and end of measurement intervals; all without heavily impacting the usage of the underlying IT infrastructure.

The KWAPI framework has been deployed as a telemetry solution in Grid5000 [47], which comprises several sites with multiple clusters. The framework collects information of both power consumption and network usage, enabling users to evaluate the performance of their experiments [46]. Even though KWAPI does not build energy consumption models itself, it *provides users and developers with feedback* on their optimisation choices and has hence been used for *profiling experiments* and building power consumption models that are incorporated in discrete-event simulators.

In addition to the default visualisation tools, which provide user feedback per experiment (*i.e.* a job or reservation), the API can be used by external systems, or scripts, to collect both measurements of instantaneous power usage and the energy consumed by specific resources over a given time frame. This information is relevant to application eco-design. Similar to monitoring network usage, the framework can *incorporate other metrics that are not energy-related.*

4.1.2. Results and Lessons Learnt

Over the past years during which the framework has been in place in Grid5000, it has addressed a list of operational challenges by providing a unified API that experimenters can use for evaluating their applications.

[4] https://wiki.openstack.org/wiki/Telemetry

[5] http://keystone.openstack.org

The functional usage of KWAPI has demonstrated several benefits of monitoring every IT resource used by large-scale applications.

Experiments have also shown that by exploiting features of monitoring hardware, KWAPI can monitor a large set of resources without perturbing experiments [46]. The throughput, latency, and jitter of measuring power consumption and transferring the measurement data have also been evaluated under several conditions [45]. The architecture and communication bus can handle significant amounts of data under short delays, but they have limitations. Under fine-grained measurement with small intervals, which can result in an enormous number of observations per second, an architecture based on stream processing systems that guarantee data processing might be more appropriate to provide users with near real-time feedback on power consumption.

Moreover, users have been increasingly interested in measuring the usage and power consumption of other types of resources, such as storage and network equipment and the temperature of multiple spots in a server room. In addition, certain users require fine-grained measurement to build and refine models of power consumption. Providing a general-purpose API that allows for such fine-grained monitoring can result in large amounts of data that need to be transferred over the network and stored for further processing. As a result, users often deploy specialised hardware and software on premises and utilise ad-hoc solutions that suit their needs. Although the KWAPI API allows for scripting and for power to be measured over the duration of an experiment, it lacks programmability functions that enable application developers to specify, at design time, which sections of code should be instrumented and the desired granularity, thus reducing the amount of collected data.

4.1.3. Ongoing and Future Work

Although KWAPI has been in use for some time on a large experimental platform, development is continuous towards incorporating other metrics to the framework, such as room and rack temperature, and other types of equipment to fulfil the needs for fine-grained measurements by certain users. Future collaborative work is also envisioned for providing programmability features to developers via existing frameworks such as EML (Section 4.2), PMLib (Section 4.3) and ECTools (Section 4.4).

4.2. Energy Measurement Library

The EML [28, 49] is a C/C++ software library built to assess the energy consumed by distributed systems. The main component, an API to speed up the measurement and experimentation process, automatically discovers available devices and monitoring capabilities while abstracting the user from hardware details.

EML was originally designed around the factory pattern and implemented in C++. Instrumented code would measure energy consumption for a supported device through a unified *Measurement Tool* interface. This design had drawbacks as it required the number and type of monitored devices to be known at compilation, and instrumentation of C code was not possible. EML has hence been redesigned to support instrumentation of C code [49]. Additional features include exporting measurement data to JSON and higher-level API methods to measure multiple types of equipment simultaneously.

4.2.1. Usage Mode

The basic usage pattern of EML used to measure a single section of code on all available devices (Figure 4) involves:

1. Library initialisation (*emlInit*), which discovers available devices and allocates necessary resources.
2. A call to *emlStart* before the relevant section of code, which may launch per-device threads that periodically perform and record measurements.
3. Calling *emlStop* after the section, which hence ends the last started monitoring interval and returns opaque *emlData_t** handles to the recorded interval data.
4. Operating on the obtained data. API functions are provided to query energy and time totals or to serialise and export data as JSON.
5. Library deinitialisation (*emlShutdown*).

The overhead introduced by EML has been evaluated and has been shown to be low [49].

4.2.2. Energy modelling with TIA and EML

Current parallel performance analysis tools are typically based on either measurement or modelling techniques, with little integration between both approaches. Researchers developing predictive models have to build their validation experiments. Conversely, most application profiling tools do not produce output that can be readily used to generate automatically approximated models.

The Tools for Instrumentation and Analysis (TIA) [50] framework was originally designed to bridge the gap between analytical complexity modelling and performance profiling tools. Through loosely

```c
#include <eml.h>
#include <stdlib.h>

int main() {
  // EML initialisation runs the device
  // discovery stage
  emlInit ();

  // get total device count and allocate
  // handles for measurement results
  size_t count ;
  emlDeviceGetCount(&count);
  emlData_t  data[count];

  // measure around a section of code and
  // retrieve results
  emlStart();
  // ... do work ...
  emlStop(data);
  // ... use data ...
  emlShutdown();
}
```

Figure 4: Minimum example of EML use.

```c
#pragma cll for(N=MIN; N<=MAX; N+=STRIDE)

/* TIA experiment: initialization */
#pragma cll init CLOCK, \
        EML_ENERGY_RAPL = init[0]*N*N
    for(i=0;i<N;i++)
      for(j=0;j<N;j++)
        A(i,j) = B(i,j) = (i == j);
#pragma cll end init

/* TIA experiment: multiplication */
#pragma cll mat CLOCK, \
        EML_ENERGY_RAPL = mat[0]*N*N*N
    for(i = 0; i < N; i++) {
      for(j = 0; j < N; j++) {
        sum = 0;
        for(k = 0; k < N; k++)
          sum += A(i, k) * B(k, j);
        C(i,j) = sum;
      }
    }
#pragma cll end mat
#pragma cll end for
```

Figure 5: Annotation of a matrix multiplication code using TIA.

```
> cll.fit(matdata, type="full",
  var.list=list("N", "N*N"))

[...]

Coefficients:
              Estimate  Std. Error  t value
(Intercept)  1.753e-01   9.206e-03    19.04
I(N * N * N) 1.338e-10   6.322e-12    21.17

Term Weights:
     weight        terms
1 0.9999998  (Intercept)
2 0.8999677        N*N*N
3 0.1000321          N*N
4 0.0000000            N

Best Top Models:
     weight                                model
1 0.8999677 EML_ENERGY_RAPL ~ I(N * N * N)
2 0.1000321      EML_ENERGY_RAPL ~ I(N * N)
```

Figure 6: Sample model for a matrix product in terms of matrix sizes.

coupled, but well- integrated components, TIA provides both profiling on a specified set of metrics for source-annotated regions of parallel code and analysis facilities to help find and validate an appropriate performance model. This methodology can also be applied to power performance, enhancing TIA with energy measurement capabilities through EML. An example of annotating code with TIA directives considering an energy-related metric is shown in Figure 5.

The data gathered from the instrumentation can be processed with an R library where a model can be obtained. Figure 6 shows an example for a matrix product in terms of matrix sizes. The computing stage has cubic complexity on the problem size, as expected. For more complex programs, the `cll.fit` function can be instructed to consider more factors and other metrics as terms. The R package includes functions to produce basic graphical visualisations of predictions and model fit graphs (Fig. 7). More sophisticated representations can take advantage of the plotting functionality in the R environment.

4.2.3. Design Goals

EML has been designed to provide an abstraction layer to different energy monitoring devices. The design is driver-oriented to ease the inclusion of new energy monitoring hardware, currently covering CPUs through RAPL interface, GPUs via NVML, Xeon Phi through MPSS and Schleifenbauer PDUs.

EML simple design can be valuable to address the challenge of *large-scale monitoring*, combined with other tools and frameworks which handle the large-scale requirements and complexity of this task. The EML interface enables instrumenting a small subset of application code as well as controlling the monitoring sample rate. It can be used as an *application profiling* tool, and combined with TIA framework *energy consumption models* can be built, including energy consumption and other performance metrics.

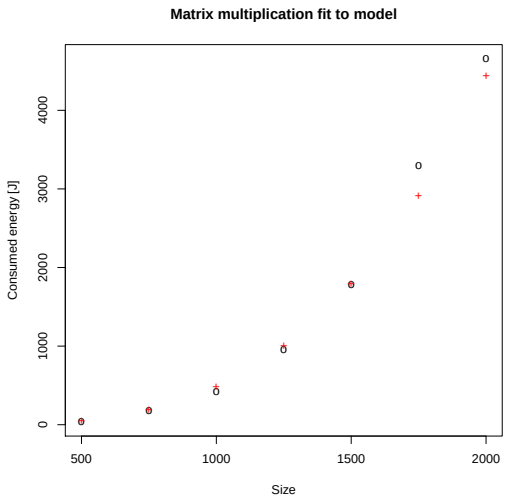

Figure 7: Generated model (red) from measurements (black)

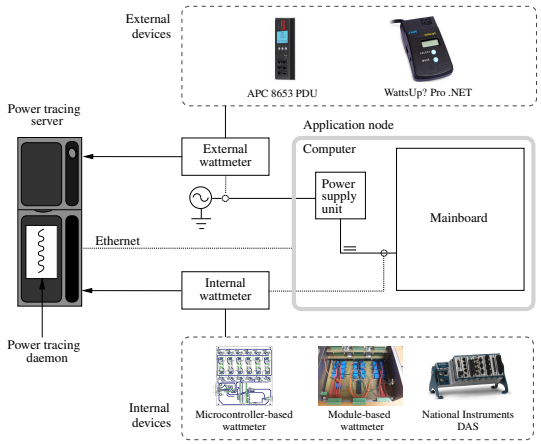

Figure 8: Architecture of PMLib.

4.2.4. *Results and Lessons Learnt*

EML has been successfully used to obtain energy analytical models [51], showing that it is viable to get structural and algorithmic parameters that affect energy consumption and efficiency. Analytical energy models have been achieved for master-slave applications [52], the High-Performance Linpack benchmark [53], and MPI communications OSU microbenchmarks [54]. From experience, we learnt that it is possible to obtain analytical models for energy consumption suitable for scheduling purposes, performance prediction, or scalability analysis. We also learnt that tuning the monitoring stage, like adjusting sampling rate, is important to obtain accurate models.

4.2.5. *Ongoing and Future Work*

Future work will comprise a production-ready modular analysis framework, an effort that involves streamlined installation and usage processes, improved visualisation and statistical capabilities, and a graphical interface for the analysis package. The EML component is also expected to undergo further API development and increase device support in coming releases.

4.3. *The* PMLib *library*

The Power Measurement Library (PMLib) is part of a framework for power-performance tracing and analysis of scientific applications [55]. This library can interface with a broad range of power measuring devices: *i)* external commercial products, such as intelligent APC PDUs, WattsUp? Pro .Net and ZES Zimmer devices; *ii)* internal DC wattmeters, like a commercial Data Acquisition System (DAS) from National Instruments (NI); and *iii)* specific designs that use microcontrollers to sample power data (see Figure 8). The complete PMLib package is publicly available [56].

Figure 9 details the framework for power-performance tracing and analysis of scientific applications. The starting point is a parallel scientific application instrumented with the PMLib software that runs on a parallel target platform – *e.g.*, a cluster, a multicore architecture, or a hybrid computer equipped with one or Graphics Processing Units (GPUs) – that consumes a certain amount of power. Connected to the target platform, there is one or several powermeter devices – either internal DC or external alternating current (AC) – that steadily monitor the power consumption, sending the data to a tracing server. Calls from the application running on the target platform to the PMLib API, instruct the tracing server to start/stop collecting the data captured by the powermeters, dump the samples in a given format into a disk file (power trace), query multiple properties of the powermeters, etc. Upon completion of the application execution, the power trace can be inspected, optionally hand-in-hand with a performance trace, using some visualization tool. Our current setting allows a smooth integration of the framework power-related traces and the performance traces obtained with Extrae. The resulting combined traces can be visualised with the Paraver tool from the Barcelona Supercomputing Centre. Nevertheless, the modular design of the framework can easily accommodate other tracing tools like, e.g., TAU, Vampir, etc.

As an additional feature, PMLib can read the C- and P-states information of each core on a multi-core plat-

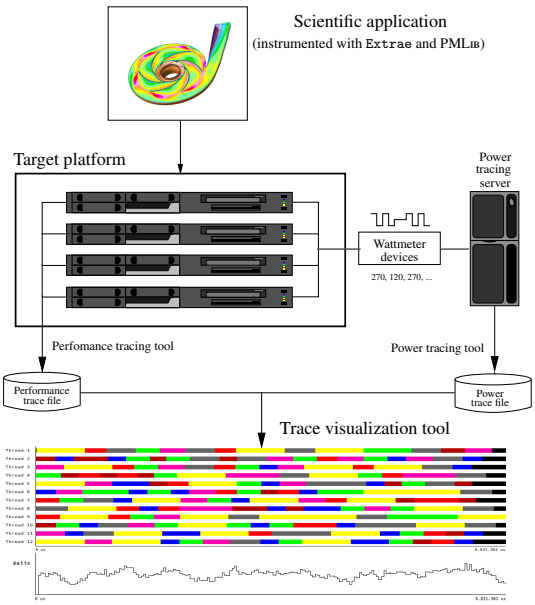

Figure 9: Collecting traces at runtime and visualization of power-performance data.

form. To do so, the server daemon integrated into the power framework reads the corresponding MSRs (Model Specific Registers) of the system, for each CPU X and state Y, with a user-configured frequency. Note that the state-recording daemon necessarily has to run on the target application and, thus, it introduces a certain overhead (regarding execution time as well as power consumption) which can become non-negligible depending on the software being monitored. The user is advised to experimentally adjust the sampling frequency of this daemon with care to avoid this effect.

4.3.1. Design Goals

For years, existing tools for HPC focused solely on monitoring and evaluation of performance metrics. Nevertheless, newer hardware has started to incorporate a broad range of sensors and measurement devices related to power consumption with varying granularity and precision. The PMLib library aims to build an integrated framework for power-performance profiling and tracing, and hence bridge the gap between performance analysis tools and power sensors. We believe that to gain a better understanding of energy usage, performance metrics, such as performance counters or everyday events, should be correlated with the power traces. Only analysing these measurements, energy inefficiencies in the code can be identified and optimised. PMLib was designed having all these needs in mind.

With PMLib we aim at facing some of the challenges above in energy monitoring. It is possible to profile and monitor USS systems with PMLib with the user interface provided by the package. To some extent, PMLib can also be used to develop power models during the training stage. Nevertheless, PMLib mainly aims to provide feedback to the users so as to gain insights about energy usage and turn them to be more energy-efficient. Finally, the modular architecture of the library allows users to create new measurement modules, either for new wattmeter devices or system sensors, such as temperature or humidity.

4.3.2. Results and Lessons Learnt

During past research, PMLib has been broadly used in several research lines: *i)* modelling power and energy consumption; *ii)* designing of energy-aware techniques for scientific applications; *iii)* developing new fine-grained power measurement devices; and *iv)* assessing accuracy of existing power measuring devices. A research result thanks to PMLib, is a tool for automatic detection of power sinks during the execution of concurrent scientific workloads [57]. This tool has been shaped in the form of a multi-threaded Python module that offers high reliability and flexibility, rendering an overall inspection process and introducing very low overheads. The detection of power sinks has been based on a comparison between the application performance trace and the C-state traces per core. When a core is supposed to be in an "idle" state but the C-state is C0 (*i.e.* active), the tool detects a power sink. Moreover, the analyser is flexible, because the task type that corresponds to "useful" work can be user-defined; furthermore, the length of the analysis interval and the divergence (discrepancy) threshold are parameters that can be adjusted by the user to the desired level. Thanks to this tool, several applications have been analysed and improved for energy efficiency.

With the development of PMLib, we learnt several lessons. While the PMLib server was implemented to support multiple types of wattmeters running concurrently with multiple users querying them, connecting or disconnecting devices from the server is sometimes not straightforward and requires to reboot the PMLib daemon running on the server. On the other hand, very long measurements at a high sampling rate may lead to memory overflows in the server. We are already aware of these shortcomings and plan to address them in future releases. A final lesson learnt is the overheads that PMLib might introduce when measuring P-/C-states. In this case, it is the user's responsibility to set the sampling frequency with care.

4.3.3. Ongoing and Future Work

While several research institutions in Spain, Germany, and France use PMLib, its development continues towards a new version of the server implemented in C++. This new version will allow using fine-grained measurement devices, with a sampling rate of several kilo-samples per second. On the other hand, we also plan to extend it to support other kinds of metrics and hardware counters available in the system. An ultimate goal will be to make the PMLib user interface compatible with the current PowerAPI from Sandia Labs.

4.4. ECTools

The Energy Consumption Library (LibEC) is an open source tool to estimate the power dissipated by a machine or an application, even if wattmeters are not available [58]. The library[6] uses a set of sensors to feed several power models. Although most information that sensors gather comes from the Linux kernel's API (*e.g.* `/sys` and `/proc` file systems), its modular design allows for the library to be easily extended to support other sources.

LibEC aims to abstract two types of sensors (*i.e.* machine-level and/or application-level sensors) and application power estimators, while providing for an extensible system where new power models can be easily integrated. The application-level sensors can be directly associated with a Process IDentification (PID) and mainly report software usage, such as performance counters. Machine-level sensors, on the other hand, report not only aggregate values for all processes but also physical property measurements that cannot be associated with a PID, such as CPU thermal dissipation. Application power estimators that are also application-level sensors comprise multiple sub-sensors used to estimate an application power consumption.

The library enables new sensors to be created and added, but here we describe those that are provided by default when explaining the types of sensors that the library supports.

Application/Machine-Level Sensors. At least one application-related variable is required to estimate the energy that an application consumes. LibEC provides PID-related sensors that gather both application and machine-level information.

As CPU, memory, and disk are often the most power-consuming server components, information on their usage is required to achieve good power models. For such one can exploit Performance Counters, CPU time, CPU elapsed time and CPU usage. Furthermore, sensors of memory usage and disk reads/writes can be used for memory and disk modelling, respectively. The sensors described here exploit PCs to obtain the counts between two updates, which are user defined.

CPU usage (CPU%) is a sensor that comprises two intermediate sensors, namely *CPU time* and *Elapsed CPU time*, that provide respectively the total CPU time – *i.e.* the sum of the system and user times – and the CPU time difference between two updates. CPU% provides the percentage of CPU that a PID or CPU core utilises. This sensor uses the elapsed CPU time divided by the machine-level CPU elapsed time, *i.e.*, the total elapsed time.

Memory usage (MEM%) is the percentage of memory that a given process uses. It collects an application's resident set size and divides it by the total available memory.

Disk Read/Write provides the number of bytes read-/written between function calls for any file partition stored on flash drives or a hard drive.

Machine-Level Sensors. In addition to application-level sensors that collect machine information related to a process, LibEC contains sensors that gather device data, such as CPU temperature, CPU frequency, and network traffic, that is not associated with a process.

The library provides interfaces to wattmeters that allow it to employ PDUs that monitor power consumption directly as well as a component that exploits ACPI information to estimate the power that a portable device consumes. The *ACPI Power Meter* retrieves the current and voltage drained by the battery from the `/sys/class/power` directory and computes its power consumption. The communication with PDUs depends on the vendor's protocol. We provide interfaces to some of Grid5000's wattmeters [47], the RECS system [60], and Energy Optimisers Limited's Plogg (an outlet adapter to measure the power dissipated by devices).

Application Power Estimators. The library enables users to integrate new power estimators. Currently, a static and two dynamic models are available to demonstrate this feature. Static models require a priori information, whereas dynamic models can auto-adapt to different workloads, but must be calibrated by a powermeter.

The simplest static model is CPU proportional, where *CPU MinMax* is a linear estimator based on the minimum (P_{min}) and maximum (P_{max}) power consumption

[6] LibEC was implemented in C++ and is distributed under the GNU General Public License (GPL) version 3.0 or later [59]

of a machine. This information is user-provided. It uses a CPU usage sensor to weight the power variance and the number of active processes ($|AP|$) to normalise the idle power (P_{min}) for each process as stated in the following equation. This estimator is architecture dependent, and its performance varies according to the accuracy of the user-provided data.

$$P_{pid} = (P_{max} - P_{min}) \times \frac{t_{cpu}^{pid}}{t_{cpu} + t_{idl}} + \frac{P_{min}}{|PR_t|}$$

When available, a wattmeter is used to achieve more precise results or to calibrate models for machines that do not have a powermeter. The *Inverse CPU power* estimator uses information on the total power consumption of a computer and attributes it to applications using a *CPU usage* sensor as stated below

$$P_{pid} = P_{PM} \times \frac{t_{cpu}^{pid}}{t_{cpu}}$$

Linear regression is a well-known method for achieving dynamic estimators by weighting pre-defined sensors and finding a model without user-provided information. The *Linear Regression Dynamic Power Estimator* estimates the weights (w_i) for any application level sensor (s_i) within the following equation:

$$P_{pid} = w_0 + \sum_{i=1}^{n} w_i * s_i$$

4.4.1. Results and Lessons Learnt

Over time, the tool has demonstrated to be a useful asset that other work can leverage. LibEC has been used to implement several solutions, such as a power monitoring tool, an application energy profiler, and dynamic power estimators. It has helped us gain insights on the advantages and limitations of monitoring power consumption and trying to correlate information with performance counters. The following describes some of the solutions based on LibEC.

Power Monitoring Tool. The Energy Consumption Monitoring Tool (ectop) is a command line application conceived to provide an easy way to monitor power estimators and compare their accuracy in real time. A user can keep track of the top consuming processes and add/remove application level sensors and power estimators.

Figure 10 shows an example of ectop with three sensors (CPU use, memory use, and disk I/O) and a power estimator (min-max CPU proportional, PE_MMC). The

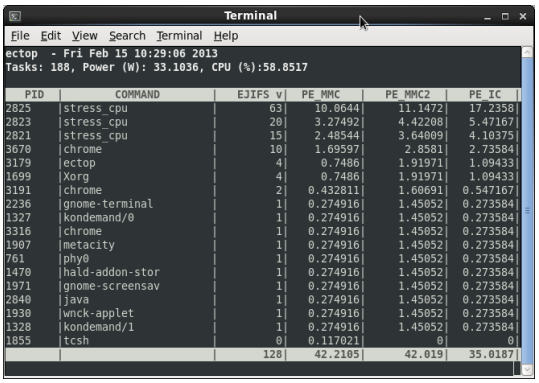

Figure 10: ectop command line interface.

ectop monitor shows a sum bar for each sensor's column, which presents its system-wide values. For the presented scenario the sum of the power estimations is larger than the value given in the power field. The power field is fed by the ACPI power estimator. This difference occurs due to the quality of the power estimator in use. As stated earlier, LibEC is a library for developing estimators that can be used for building other models.

Application Energy Profiling. Valgreen is an application energy profiler [61] that uses LibEC to assist a programmer in implementing energy efficient algorithms by sampling the power consumption of a given application in small time steps. These time steps may be configured; the smaller the step, the more precise the energy measurement.

Dynamic Power Estimators. Auto-generated models have been widely used for application power estimation. This use case compares a dynamic model that is updated through linear regression in regular time intervals with a simple static model. These dynamic models do not need any user input and can run with different devices. The idea is to show the importance of a dynamic model when the workload on the machine changes and our model is no longer valid.

Figure 11 presents a comparison of a dynamic model, the actual power measured with a wattmeter, and a static model configured in another machine. Over time the adaptive model approaches to the actual dissipated power while the static one diverges. Here we show the total power consumption of the machine, but the same goes with the power consumed by each process.

4.4.2. Ongoing and Future Work

Future work on ectop library includes the integration of new dynamic power models, in particular, those con-

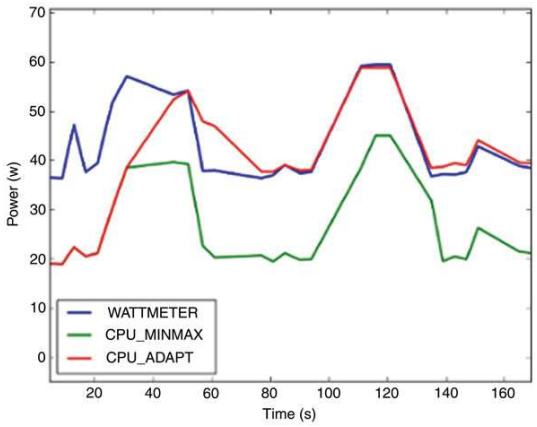

Figure 11: Comparison between a wattmeter, an adaptive model (CPU_ADAPT) and a static model (CPU_MINMAX).

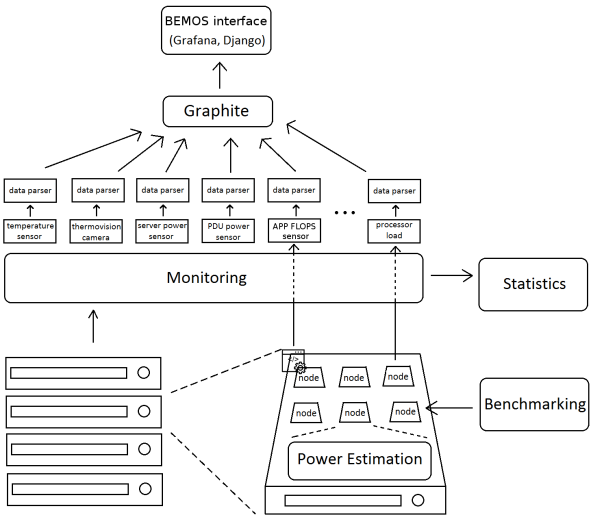

Figure 12: The architecture of the BEMOS platform.

structed using machine learning techniques [62]. We also plan to integrate other wattmeters, probably by interfacing with other tools such as the Kwapi presented in Section 4.1.

4.5. BEMOS

The Energy Benchmarking, Monitoring and Statistics System (BEMOS) is a platform designed by Poznan Supercomputing and Networking Center to manage and monitor large-scale heterogeneous server infrastructure and environmental parameters of a server room. The system integrates sensors from multiple tools, which are synchronised with the BEMOS Network Time Protocol (NTP) server to ensure data consistency over time.

The system architecture, depicted in Figure 12, relies on a Graphite backend server [63]. A circular-buffer database, similar in design to RRDs, stores the time-series data collected from multiple types of sensors. The database assumes a pre-specified number of stored values thus "wrapping around" after some time to overwrite the oldest data. The values are also aggregated over time, trading accuracy for collecting data over longer periods of time. Values from each sensor are stored in its database file. Their size is specified before the beginning of the experiments and does not change.

The solution is designed to handle large numbers of numeric time-series data, like dozens of performance metrics from thousands of servers. It scales horizontally and is fault tolerant, avoiding a single point of failure. Data aggregation does not allow for calculating accurate metrics over extended periods of time. Therefore, the Statistics module was designed to collect data over pre-specified time periods and store it in a relational database. Statistics and graphs can be created from this data using R environment [64] before the data is sent to the Graphite server.

In addition to monitoring, BEMOS is used for server management; *e.g.*, executing benchmarks on servers for further monitoring. The management module allows creating configuration files that define the automatic execution of specific applications and servers on which they run, sequentially or in parallel.

The system allows collecting information from sensors of multiple types and producers. An abstraction layer between sensors and the database allows independence from sensor implementation details. A module for obtaining measurements must be created to pool the sensor and send the data to the Graphite service. Sensors are not necessarily physical devices. A software-defined sensor can monitor various metrics including processor load, the number of active users, parameters of running applications such as FLOPS database transactions. Thus, BEMOS also enables monitoring applications and services.

Measurements from sensors can be taken at configurable time intervals and granularity. The sensors currently deployed in our infrastructure take measurements every 1 or 5 seconds, depending on the user requirements. The system also allows for specifying rules with actions to execute if certain anomalies arise (*e.g.* send an e-mail if the temperature exceeds a given threshold).

A module for estimating the power server consumes [65] enables gathering power data of servers without wattmeters or servers with low-precision wattmeters. The module can calculate the power usage of a ma-

chine under any load at runtime based on the analysis of performance counters. Training data is used for a given server to create a model. The execution of pre-selected applications is monitored under multiple conditions, during which the following parameters are observed: CPU frequency; P-State voltage table; values from multiple CPU-related performance counters (PeC) such as number of instructions, cycles, branches and CPU cache counters including loads, load misses, stores, pre-fetches; and memory-related counters such as number of performed bus transactions. Power usage values of the whole server are retrieved simultaneously. The power meter is only used once to collect the training data.

The model is created using linear regression where the accuracy criterion is the adjusted R-squared value. As many variables show non-linear correlation with machine power consumption, an appropriate transformation is automatically established so that linear regression could use these variables. The algorithm chooses among no transformation, logarithmic, and square root. The choice is made based on the comparison of R-squared values of models with different transformations applied to the analysed variable. Once the model is created, it can serve as a sensor in the BEMOS platform and be used to estimate in real-time the power consumed by a server under any load. The overhead on the processor resulted from calculating the values for the model is negligible.

Results are presented either directly in the interactive graphical interface accessible via Web browser and based on the Django framework. It provides interactive, easily editable graphs, with the possibility to extensively adapt charts for user's purposes. The combination of Graphite with Grafana also gives the opportunity to manipulate the collected series with a set of predefined functions, for instance taking the average of multiple series, or the derivative of one of the series concerning time. This opportunity enables performing fast and easy prototyping and seeking for patterns and dependencies within the collected datasets.

4.5.1. Design Goals

BEMOS addresses the following main challenges related to monitoring large-scale systems:

- It combines computing systems' power usage measurement data with infrastructure measurements and environmental parameters such as temperature, air flow, humidity, and others.
- It can efficiently monitor multi-layered systems (with different levels of detail) that generate large amounts of monitoring data with varying accuracy.
- It enables monitoring computing systems where the density is too high to install individual sensors and collect precise information.
- It allows to disaggregate coarse-grained data into smaller, more detailed information, *e.g.* by defining virtual temperature sensors using an infrared camera, and by power usage estimations based on application characteristics.

4.5.2. Results and Lessons Learnt

BEMOS system was successfully deployed at Poznan Supercomputing and Networking Center to gather energy data from a large number of sensors. The available sensors are currently the following:

- Sensors at the level of a single server: processor load, processor/GPU temperature, processor/memory/GPU power, memory usage, network card usage, disk usage, air temperature, the velocity at the air inlet and outlet, temperature of the liquid in case of liquid-cooled servers.
- Sensors at the level of the room: temperature of the air from the air conditioner, the temperature of the room, the power drawn by all servers, humidity of the air in the room.
- Thermal imaging camera that allows monitoring larger areas, such as multiple servers; at the same time, it is possible to accurately select specific points in the thermal image and retrieve values of temperature from them over time.
- Power data from intelligent PDUs that collect consumption information from multiple connected servers.

The accuracy of the Estimation module was also analysed. Based on the experiments performed on some commodity servers available on the market despite the architectural differences of evaluated machines and various characteristics of tested applications, the results were very stable and reliable, with the mean square error within 4% [65]. Figure 13 presents the real and estimated power consumption of a server while running Make application on an Intel processor.

From these experiences, we learnt that the presented system is capable of efficiently collecting significant amounts of data from heterogeneous sensors. The drawback of this solution is that more detailed information is lost over longer periods of time, something that

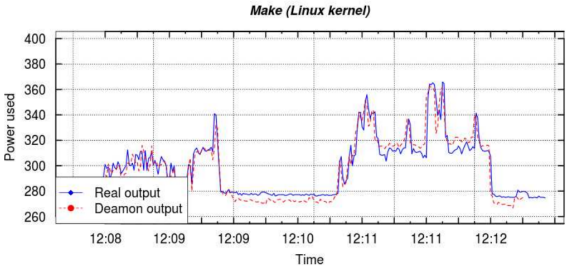

Figure 13: Power consumption estimation while running Make application.

the Statistics module tries to compensate by enabling deeper analyses over shorter executions.

4.5.3. Ongoing and Future Work

Future work will consist in implementing further capabilities regarding intelligent management of analysed resources. Based on collected data the system may take automatic decisions to reduce energy consumption further.

4.6. Summary of the solutions

In this section, we examine the aforementioned solutions for energy monitoring of large-scale computing systems and review which challenges each solution addresses. This review aims at comparing functionality supported by the tools so as to know their features and to give a general overview of the design goals and trends followed.

As shown in Table 3, most challenges stated in Section 3 are either fully or partially addressed by the tools. Regarding challenges **C1** and **C3**, we observe that they are moderately tackled by all tools at different levels. Hence, supporting large-scale profiling and monitoring and providing feedback to the end-user are important requirements. Nevertheless, for challenge **C1** we notice that some of the tools do not offer support for profiling applications (*C1.2-C1.3*), and services are not fully scalable (*C1.4*). Similarly, challenge **C3** is not fully addressed by all solutions, as some of them do not offer interactive graphical interfaces (*C3.2*). Focusing on challenge **C2**, we detect that providing models for simulating energy consumption was not directly tackled by the tools (*C2.1*), but they can be integrated with simulation tools (*C2.2*). The main reason is that models go one step ahead of power monitoring as they first require profiling and tracing capabilities. Therefore, KWAPI, EML and PMLib, need extra solutions or frameworks to derive proper power and energy models. Finally, we detect that challenge **C4** was addressed by all tools. Thanks to their modular design, extending these tools with other measurement sources does not imply huge efforts.

5. Gap Analysis and Future Directions

This section first discusses two large-scale system scenarios and then describes a list of topics that deserve attention from the research community. The first scenario illustrates the needs of large Hadoop setups that mostly run on data centres and clouds. As discussed in previous work [3], unification of HPC and big data analytics is required to address challenges in data processing in major research domains. The second scenario considers highly distributed services that, in addition to services hosted at traditional data centres and clouds, are increasingly making use of the Internet edges; something often referred to as edge or fog computing.

5.1. Monitoring needs of Hadoop Ultrascale Systems

MapReduce [66] is a popular programming model employed for processing large amounts of data, either on premises or on the cloud. Apache Hadoop, an open-source MapReduce implementation designed to handle hundreds of TeraBytes (TBs), works by splitting the data into small chunks that can be processed in parallel using thousands of processor cores. Hadoop also provides a simplified model for developing distributed applications based on Map and Reduce phases and mechanisms for distributed data management and fault tolerance.

Many large organisations employ Hadoop on a daily basis to manage their ultrascale datasets. Due to the size of the ultrascale systems and the unique features of Hadoop, monitoring of power consumption is not a trivial task. Ideally, the energy consumed by all Hadoop nodes should be measured, but this is tough for ultrascale systems – especially with real wattmeters. To effectively monitor the energy consumption, software and hardware must be combined. This combination could lead to more sophisticated and powerful monitoring tools, which take into consideration more advanced features such as the homogeneity of the cluster and, as a result, achieve overhead reduction and production of a manageable amount of data.

To improve the energy efficiency of Hadoop ultrascale systems, energy monitoring tools also have to support a variety of measurements from a low level (*e.g.* CPU, RAM) to a high level (a group of nodes or the whole cluster), with a low or high sampling frequency.

Table 3: Challenges addressed by the solutions.

Challenges	KWAPI	EML	PMLib	ECTools	BEMOS
C1: Large-scale profiling and monitoring	✓	✓	✓	✓	✓
C1.1: Fine-grained energy monitoring	✓	✓	✓	✓	✓
C1.2: Application and service monitoring	✗	✗	✓	✓	✓
C1.3: Resource and service profiling	(✓) – resource profiling	✗	(✓) – resource profiling	✗	✗
C1.4: Scalable architectures	✓	✗	✗	✗	✗
C2: Models and simulation of energy consumption	✗	(✓) – Yes, but requires TIA framework	(✓) – Yes, but requires extra tools	✓– It comes with a few models to estimate power	✓
C2.1: Models and estimations	✗	✗	✓	✓	✓
C2.2: Integration with simulation tools	✗	✓	✓	✓	✓
C3: APIs and developer/user feedback	✓	✓	✓	✓	✓
C3.1: APIs	✓	✓	✓	✓	✓
C3.2: Interactive graphical interfaces	✗	✗	✗	✓	✓
C4: Integrating other measurements in architectures and APIs	✓	✓	✓	✓	✓

It is quite possible to need measurements from a cluster over several days to find how Hadoop characteristics affect the energy consumption in the long term, or measurements per node per second to analyse energy consumption of map and reduce phases. Furthermore, except for the energy consumption of the nodes, there is the need for monitoring network devices such as switches and physical systems such as air conditioning and lighting.

5.2. Massively Distributed Systems

A large part of the "big data" produced today is received by processing tools in near real time and is most important when analysed quickly. Under application scenarios, such as smart cities, operational monitoring of large infrastructures, Internet of Things [67], and massively distributed games, continuous streams of data must be processed under short delays. Existing solutions to such scenarios often comprise multiple tiers, such as:

- Push/pull APIs and underlying systems for collecting data from monitoring devices or for graphical interfaces to provide end-user input.
- Querying, buffering, and publish/subscribe systems to transfer data to data centres or intermediate locations such as micro data centres located closer to end users.
- Stream processing systems that carry out data processing using a dataflow abstraction or in micro-batches.
- Highly distributed NoSQL/SQL storage solutions to store data to be processed by batch systems such as Hadoop and in-memory processing solutions.
- Interactive visualisation frameworks that enable users to start and stop queries that can dynamically impact the amount of processing required by the analysis tools.

Although building and managing such systems present their own challenges about scalability, they often rely on complex hardware and software infrastructure that may not always be offered by a single provider. Moreover, the system itself presents requirements of near-realtime processing that cannot be compromised by heavyweight monitoring solutions. Energy consumption accounts for a significant share of the operational cost of these systems, but monitoring and optimising the energy consumed by their components is not trivial.

5.3. Issues with Current Monitoring Solutions

Most of the discussed solutions address energy monitoring requirements at data centres and large compute clusters. As systems grow in scale, complexity and heterogeneity, it becomes a challenge to build scalable

monitoring solutions that can cope with the amount of data collected without introducing significant overheads. Some of the areas in which the current systems still encounter difficulty include:

- **Scalability** – although existing solutions can handle fairly large clusters, monitoring a wide range of metrics with per-second granularity, systems that can achieve the Level 3 of the EE HPC methodology presented in Section 2.2 are still the exception. Ultrascale systems can pose additional scalability issues to solutions that already struggle to handle current scenarios.

- **Monitoring overhead** – monitoring solutions should not incur significant costs on the monitored system. Although efforts have been made to reduce the overhead introduced my energy monitoring of large clusters, as the software systems increasingly present near-realtime processing requirements, the overhead introduced by existing monitoring solutions can become an issue.

- **Data management** – similar to many systems that have to handle large amounts of data, energy monitoring can become a so-called big-data system as not only power consumption information is collected, but also information on resource usage and other variables with which information is correlated during analysis. This scenario can be exacerbated by the fact that administrators, engineers, and decision makers are increasingly willing to perform *predictive* and *prescriptive* data analyses using interactive interfaces. These analyses can require adjusting data collection and monitoring metrics on demand. Current solutions fall short of supporting such use patterns.

- **Scalable architectures** – monitoring solutions should be able to scale horizontally, allowing administrators to add more capacity on demand. Existing solutions try to cope with scalability challenges and responsiveness by adding dedicated hardware to host and execute the monitoring solution. The size of such dedicated infrastructure is often statically defined based on the expected monitoring needs, which may, in turn, make the monitoring system a large energy consumer. Architectures that can cope with the monitoring load and adjust allocated capacity on demand are preferable, but still not an ordinary reality.

- **Power consumption models** – the use of power models can help reduce the number of required monitored devices and sample rates when resources across a cluster are statistically similar, which is the case of several Hadoop settings. However, as the heterogeneity of large-scale systems grows and services make increasing use of equipment at the Internet edge, it becomes more complicated to build accurate energy models for a large spectrum of devices. Moreover, many software systems are deployed on virtual machines and lightweight containers, whose individual energy consumption is often difficult to determine. Present solutions do not offer systematic means to build and include such models in monitoring systems.

5.4. Future Directions

The measurement and monitoring overhead should be kept to a bare minimum to cope with ultrascale systems. Application and hardware energy profiles should also scale to millions of cores and consider large virtualised infrastructures that make use of techniques such as lightweight containers. As previously mentioned, monitoring systems could exploit specialised hardware (*e.g.* FPGAs) to reduce the delay of gathering data and making it available to APIs and visualisation interfaces. In addition to specialised equipment, monitoring architectures could exploit lightweight virtualisation to achieve horizontal scalability and elasticity, avoiding over-provisioning compute resources to monitor tasks.

Flexible programming techniques that can enable developers to adjust the monitoring granularity and intervals dynamically for specific periods depending on the monitoring system workload can help reduce its overhead and improve scalability. Also concerning architectures, scalability and data management can be enhanced by employing multilayered approaches where individual elements are responsible for a subset of the infrastructure. Such method can prevent transferring large amounts of monitoring information across the network still storing the data if required.

Work carried out by designers of monitoring systems, and visualisation should be intertwined to allow for adjusting data collection and management processes dynamically, based on insights that a user or service designer may gain while interacting with a visualisation tool. At present, there is still a gap between the two communities that should be bridged. We also envision an increasing use of machine learning techniques, such as deep learning [68], not only for understanding how a large system consumes energy but also for building sophisticated models of power consumption and for managing ultrascale systems.

6. Conclusions

As USSs join parallel and distributed computing systems that will be two to three orders of magnitude larger than today's infrastructure, fine-grained monitoring of the power consumed by their underlying IT resources can be challenging. In this work, we surveyed state of the art on energy monitoring of large-scale systems and methodologies for monitoring the power consumed by large systems. We discussed some of the challenges towards monitoring and improving the energy efficiency of USSs.

Previous work often extrapolates energy measurements of large systems. Extrapolations can lead to variations of up to 20% due to measurement timing and 10-15% due to insufficient sample sizes [36]. There is a need for high-quality levels of measurement and power-monitoring infrastructure that can provide timely and accurate feedback to system developers and application writers so that they can optimise the energy use [41].

Many challenges must be addressed to significantly reduce the energy consumption and enable fine-grained measurement in complex and large-scale systems. Appropriate architectures must be investigated to allow monitoring to provide sufficiently detailed information and avoid communication bottlenecks. This architecture should enable, through carefully designed communication and interfaces, accessing fine-grained measurement data when and where needed. As the use of hardware measurement devices for all components of USS could cause communication bottlenecks and high additional costs, accurate estimation methods would be very useful. The overall energy consumption also depends on environmental parameters such as temperature, air/liquid flow, humidity, cooling system power usage, among other factors. Hence, effective integration of these measurements is an important part of monitoring.

Acknowledgements

This work was carried out by members of the Energy Efficiency Working Group of the Nesus European COST Action IC1305. It was partly funded by the European Commission under contract 288701 through the project CoolEmAll and supported by the French Fonds national pour la Société Numérique (FSN) XLCloud project. Some results discussed in this paper were carried out using the Grid5000 experimental testbed, being developed under the Inria ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://grid5000.fr). This work was also partly funded by FWF through project Y 904 START-Program 2015.

References

[1] M. D. de Assuncao, R. N. Calheiros, S. Bianchi, M. A. S. Netto, R. Buyya, Big data computing and clouds: Trends and future directions, Journal of Parallel and Distributed Computing 79–80 (0) (2015) 3–15. doi:http://dx.doi.org/10.1016/j.jpdc.2014.08.003.

[2] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, et al., Exascale computing study: Technology challenges in achieving exascale systems, Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep 15.

[3] D. A. Reed, J. Dongarra, Exascale computing and big data, Communications of the ACM 58 (7) (2015) 56–68. doi:10.1145/2699414.

[4] L. A. Barroso, U. Hölzle, The case for energy-proportional computing, Computer 40 (12) (2007) 33–37. doi:10.1109/MC.2007.443.

[5] A.-C. Orgerie, M. D. d. Assuncao, L. Lefevre, A survey on techniques for improving the energy efficiency of large-scale distributed systems, ACM Computing Surveys (CSUR) 46 (4) (2014) 47.

[6] A. Kulseitova, A. T. Fong, A survey of energy-efficient techniques in cloud data centers, in: ICT for Smart Society (ICISS), 2013 International Conference on, IEEE, 2013, pp. 1–5.

[7] R. Bolla, R. Bruschi, F. Davoli, F. Cucchietti, Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures, IEEE Communications Surveys Tutorials 13 (2) (2011) 223–244. doi:10.1109/SURV.2011.071410.00073.

[8] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Software: Practice and Experience 32 (2) (2002) 135–164.

[9] M. Bagein, J. Barbosa, V. Blanco, I. Brandic, S. Cremer, S. Fremal, H. Karatza, L. Lefevre, T. Mastelic, A. Oleksiak, A.-C. Orgerie, G. Stavrinides, S. Varrette, Energy efficiency for ultrascale systems: Challenges and trends from nesus project, Supercomputing Frontiers and Innovations 2 (2) (2015) 105–131.
URL http://superfri.org/superfri/article/view/48

[10] S. Benedict, Energy-aware performance analysis methodologies for HPC architectures – an exploratory study, Journal of Network and Computer Applications 35 (6) (2012) 1709–1719.

[11] A. Noureddine, R. Rouvoy, L. Seinturier, A review of energy measurement approaches, SIGOPS Oper. Syst. Rev. 47 (3) (2013) 42–49.

[12] C.-H. Hsu, S. W. Poole, Power measurement for high performance computing: State of the art, in: International Green Computing Conference and Workshops (IGCC), 2011, pp. 1–6.

[13] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, W. E. Nagel, Power measurement techniques on standard compute nodes: A quantitative comparison, in: International Symposium on Performance Analysis of Systems and Software (ISPASS), 2013, pp. 194–204.

[14] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, A. V. Vasilakos, Cloud computing: Survey on energy efficiency, ACM Computing Surveys 47 (2) (2014) 33:1–33:36.

[15] P3 international: Kill a watt product page, http://www.p3international.com/products/p4400.html.

[16] Watts up meters: Watt's up product page, https://www.wattsupmeters.com/.
[17] R. Ge, X. Feng, S. Song, H. C. Chang, D. Li, K. W. Cameron, Powerpack: Energy profiling and analysis of high-performance systems and applications, IEEE Transactions on Parallel and Distributed Systems 21 (5) (2010) 658–671.
[18] J. Flinn, M. Satyanarayanan, Powerscope: a tool for profiling the energy usage of mobile applications, in: 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99), 1999, pp. 2–10.
[19] G. Da Costa, J.-M. Pierson, L. Fontoura Cupertino, Mastering system and power measures for servers in datacenter, Sustainable Computing: Informatics and Systems 15 (2017) 28–38.
URL http://dx.doi.org/10.1016/j.suscom.2017.05.003
[20] D. Bedard, M. Y. Lim, R. Fowler, A. Porterfield, Powermon: Fine-grained and integrated power monitoring for commodity computer systems, in: IEEE SoutheastCon 2010 (SoutheastCon), 2010, pp. 479–484.
[21] J. H. Laros, P. Pokorny, D. DeBonis, Powerinsight – a commodity power measurement capability, in: International Green Computing Conference (IGCC), 2013, pp. 1–6.
[22] Arm limited: Arm energy probe, http://ds.arm.com/ds-5/optimize/arm-energy-probe/.
[23] S. Ryffel, Lea2p–the linux energy attribution and accounting platform, Master's thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.
[24] D. Hackenberg, T. Ilsche, J. Schuchart, R. Schöne, W. E. Nagel, M. Simon, Y. Georgiou, Hdeem: High definition energy efficiency monitoring, in: International Workshop on Energy Efficient Supercomputing (E2SC), IEEE Press, 2014. doi:10.1109/e2sc.2014.13.
[25] Linux man-pages project: perf event open(2) (february 2013).
[26] S. Browne, J. Dongarra, N. Garner, G. Ho, P. Mucci, A portable programming interface for performance evaluation on modern processors, International Journal of High Performance Computing Applications 14 (3) (2000) 189–204.
[27] Sandia national laboratories: High performance computing power application programming interface (API) specification, http://powerapi.sandia.gov/.
[28] A. Cabrera, F. Almeida, J. Arteaga, V. Blanco, Energy measurement library (eml) usage and overhead analysis, in: 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015, pp. 554–558. doi:10.1109/PDP.2015.120.
[29] HDEEM Library Reference Guide, Tech. rep., Bull Atos Technologies (2016).
[30] Y. Georgiou, T. Cadeau, D. Glesser, D. Auble, M. Jette, M. Hautreux, Energy accounting and control with SLURM resource and job management system, in: Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings, 2014, pp. 96–118.
[31] Spec power and performance benchmark methodology v2.1, https://www.spec.org/power/.
[32] D. Hackenberg, R. Schöne, D. Molka, M. S. Müller, A. Knüpfer, Quantifying power consumption variations of HPC systems using SPEC MPI benchmarks, Computer Science – Research and Development 25 (3) (2010) 155–163.
[33] T. R. Scogland, C. P. Steffen, T. Wilde, F. Parent, S. Coghlan, N. Bates, W.-c. Feng, E. Strohmaier, A power-measurement methodology for large-scale, high-performance computing, in: 5th ACM/SPEC International Conference on Performance Engineering, ICPE '14, ACM, New York, USA, 2014, pp. 149–159. doi:10.1145/2568088.2576795.
URL http://doi.acm.org/10.1145/2568088.2576795
[34] http://www.top500.org/.
[35] http://www.green500.org/.
[36] T. Scogland, J. Azose, D. Rohr, S. Rivoire, N. Bates, D. Hackenberg, Node variability in large-scale power measurements: Perspectives from the green500, top500 and eehpcwg, in: International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, ACM, New York, USA, 2015, pp. 74:1–74:11.
[37] M. E. M. Diouri, O. Glück, L. Lefèvre, J.-C. Mignot, Your Cluster is not Power Homogeneous: Take Care when Designing Green Schedulers!, in: IGCC - 4th IEEE International Green Computing Conference, Arlington, United States, 2013. doi:10.1109/IGCC.2013.6604506.
URL https://hal.inria.fr/hal-00870637
[38] M. E. M. Diouri, M. F. Dolz, O. Glck, L. Lefvre, P. Alonso, S. Cataln, R. Mayo, E. S. Quintana-Ort, Assessing power monitoring approaches for energy and power analysis of computers, Sustainable Computing: Informatics and Systems 4 (2) (2014) 68 – 82, special Issue on Selected papers from EE-LSDS2013 Conference.
[39] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, Mobile edge computing: A key technology towards 5G, Whitepaper ETSI White Paper No. 11, European Telecommunications Standards Institute (ETSI) (September 2015).
[40] J. Mair, Z. Huang, D. Eyers, Manila: Using a densely populated pmc-space for power modelling within large-scale systems, in: 45th International Conference on Parallel Processing Workshops (ICPPW 2016), 2016, pp. 210–219. doi:10.1109/ICPPW.2016.41.
[41] K. Yoshii, K. Iskra, R. Gupta, P. Beckman, V. Vishwanath, C. Yu, S. Coghlan, Evaluating power-monitoring capabilities on ibm blue gene/p and blue gene/q, in: 2012 IEEE International Conference on Cluster Computing, 2012, pp. 36–44.
[42] J. Davey, F. Mansmann, J. Kohlhammer, D. Keim, The future internet, Springer-Verlag, Berlin, Heidelberg, 2012, Ch. Visual Analytics: Towards Intelligent Interactive Internet and Security Solutions, pp. 93–104.
[43] R. Kosara, J. Mackinlay, Storytelling: The Next Step for Visualization, Computer 46 (5) (2013) 44–50.
[44] D. Fisher, R. DeLine, M. Czerwinski, S. Drucker, Interactions with Big Data Analytics, Interactions 19 (3) (2012) 50–59.
[45] F. Rossigneux, L. Lefevre, J.-P. Gelas, M. Dias De Assuncao, A Generic and Extensible Framework for Monitoring Energy Consumption of OpenStack Clouds, in: The 4th IEEE International Conference on Sustainable Computing and Communications (Sustaincom 2014), Sydney, Australia, 2014.
URL https://hal.inria.fr/hal-01094387
[46] F. Clouet, S. Delamare, J.-P. Gelas, L. Lefèvre, L. Nussbaum, C. Parisot, L. Pouilloux, F. Rossigneux, A Unified Monitoring Framework for Energy Consumption and Network Traffic, in: TRIDENTCOM - International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities, Vancouver, Canada, 2015, p. 10.
URL https://hal.inria.fr/hal-01167915
[47] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lantéri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, T. Iréa, Grid'5000: a large scale and highly reconfigurable experimental Grid testbed, Int. Journal of High Performance Computing Applications 20 (4) (2006) 481–494.
[48] P. Hintjens, ZeroMQ: Messaging for Many Applications, O'Reilly Media, Sebastopol, USA, 2013.
[49] F. Almeida, J. Arteaga, V. Blanco, A. Cabrera, Energy measurement tools for ultrascale computing: A survey, Supercomputing

Frontiers and Innovations 2 (2).

[50] D. Martinez, J. Cabaleiro, T. Pena, F. Rivera, V. Blanco, Accurate analytical performance model of communications in MPI applications, in: 23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009, 2009, pp. 1–8. doi:10.1109/IPDPS.2009.5161175.
URL http://dx.doi.org/10.1109/IPDPS.2009.5161175

[51] A. C. Pérez, F. Almeida, J. Arteaga, V. B. Pérez, Measuring energy consumption using EML (energy measurement library), Computer Science - R&D 30 (2) (2015) 135–143. doi:10.1007/s00450-014-0269-5.
URL http://dx.doi.org/10.1007/s00450-014-0269-5

[52] F. Almeida, V. B. Pérez, A. C. Pérez, J. Ruiz, Modeling energy consumption for master-slave applications, The Journal of Supercomputing 65 (3) (2013) 1137–1149.

[53] A. C. Pérez, F. Almeida, V. B. Pérez, D. Giménez, Analytical modeling of the energy consumption for the high performance linpack, in: PDP, IEEE Computer Society, 2013, pp. 343–350.

[54] F. Almeida, V. B. Pérez, I. Gonzalez, A. C. Pérez, D. Giménez, Analytical energy models for MPI communications on a sandybridge architecture, in: ICPP, IEEE Computer Society, 2013, pp. 868–876.

[55] S. Barrachina, M. Barreda, S. Catalán, M. F. Dolz, G. Fabregat, R. Mayo, E. S. Quintana-Ortí, An integrated framework for power-performance analysis of parallel scientific workloads, 2013, pp. 114–119.

[56] M. F. Dolz, https://github.com/mdolz/PMLib.

[57] M. Barreda, S. Catalán, M. F. Dolz, R. Mayo, E. S. Quintana-Ortí, Automatic detection of power bottlenecks in parallel scientific applications, Fourth International Conference on Energy-Aware High Performance Computing, Computer Science Research and Development 29(3-4) (2013) 221–229.

[58] L. F. Cupertino, G. D. Costa, A. Sayah, J.-M. Pierson, Energy Consumption Library, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 51–57.

[59] Cupertino, l.f.: Energy consumption tools web-page (https://www.irit.fr/georges.da-costa/ectools.tgz) (april 2013).

[60] Christmann: Description for resource efficient computing system (recs) (2009).

[61] L. Fontoura Cupertino, G. Da Costa, A. Sayah, J.-M. Pierson, Valgreen: an Application's Energy Profiler, International Journal of Soft Computing and Software Engineering, SCSE'13 conference 3 (3).
URL http://dx.doi.org/10.7321/jscse.v3.n3.83

[62] L. Fontoura Cupertino, Modeling the power consumption of computing systems and applications through Machine Learning techniques, Phd thesis, Universit Paul Sabatier, Toulouse, France, (Defence on July 17, 2015) (July 2015).

[63] https://github.com/graphite-project/graphite-web.

[64] https://www.r-project.org/.

[65] M. Jarus, A. Oleksiak, T. Piontek, J. Weglarz, Runtime power usage estimation of HPC servers for various classes of real-life applications, Future Generation Computer Systems 26 (2014) 299–310.

[66] http://hadoop.apache.org/.

[67] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Computer Networks 54 (15) (2010) 2787–2805.

[68] I. Arel, D. C. Rose, T. P. Karnowski, Deep machine learning – a new frontier in artificial intelligence research, IEEE Computational Intelligence Magazine 5 (4) (2010) 13–18.